In [1]:
```
#  By Symon Kimitei
#  Linear Regression
#  March 10th, 2021
#---------------------------------------------------
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os

#========================================================================
# Download the dataset
os.chdir("C:/Users/kimit/OneDrive/Desktop/py/hw2")

# read in the train data set into pandas dataframes and add a constant
# as a predictor in the training data set
train_df = pd.read_csv("lr_training.csv") # x
train_X = train_df.drop("label",axis=1)
train_X["Constant"] = 1 # constant for b0
train_y = train_df.label # y

train_X.head(5)
```

Out[1]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel77
0	0	0	0	0	0	0	0	0	0	0	...	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 785 columns

In [2]:
```
# read in the test data set into pandas dataframes and add a constant
# as a predictor in the test data set
test_df = pd.read_csv('lr_test.csv')
test_X = test_df.drop("label",axis=1) # x
test_y = test_df.label # y
test_X['Constant'] = 1 # constant b0

test_X.head(5)
```

Out[2]:

	pixel0	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	...	pixel775	pixel77
0	0	0	0	0	0	0	0	0	0	0	...	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	

5 rows × 785 columns

In [3]:
```
### Solving by using the OLS algorithm ###
#------------------------------------------------------------
# Derive the beta values utilizing the closed form solution

# the calculation of the optimal beta weights using linear algebra
b_opt = np.linalg.pinv(train_X.T.dot(train_X)).dot(train_X.T).dot(train_y)
b_opt
```

Out[3]:
```
array([ 0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
        0.00000000e+00,  0.00000000e+00,  0.00000000e+00,  1.72017169e-05,
        1.72017169e-05, -5.01671304e-16, -1.41248856e-15, -3.59899645e-16,
       -8.00426751e-16, -8.28872157e-16,  1.56147644e-15,  4.35313949e-17,
        1.04681003e-15,  6.02499778e-16,  8.11173965e-17,  5.67073103e-16,
```

In [5]:
```
# using the beta weights and data to predict y hat (train)
train_yhat = train_X.dot(b_opt)

# the cost function
def cost(yhat,y):
    return np.mean((yhat - train_y)**2)

# calculates the cost function for X train
print(cost(train_yhat,train_y))
```

```
3.476328185145509e-27
```

In [6]:
```
# using the beta weights and data to predict y hat (test)
test_yhat = test_X.dot(b_opt)

# calculates the cost function for X test
print(cost(test_yhat,test_y))
```

```
0.5263540282257697
```

In [18]:

```
# classifying all predicted values above .5 as 1 and below .5 as 0 (train)
train_yclass = (train_X.dot(b_opt) > .5).values

# prints out the coefficients:
coefficients_LR =[]
for (pixel,coef) in dict(zip(train_X.columns,b_opt)).items():
    coefficients_LR.append(coef)

# finds the proportion of correctly predicted classes (train)
print("Training Classification Accuracy:",np.mean(train_yclass == train_y))

# classifying all predicted values above .5 as 1 and below .5 as 0 (test)
test_yclass = (test_X.dot(b_opt) > .5).values

# finds the proportion of correctly predicted classes (train)
print("Test Classification Accuracy:",np.mean(test_yclass == test_y))

coefficients_LR=pd.DataFrame(coefficients_LR)
coefficients_LR.columns=["Coefficients"]
coefficients_LR.index.names = ['Pixel number']
coefficients_LR
```

```
Training Classification Accuracy: 1.0
Test Classification Accuracy: 0.98
```

Out[18]:

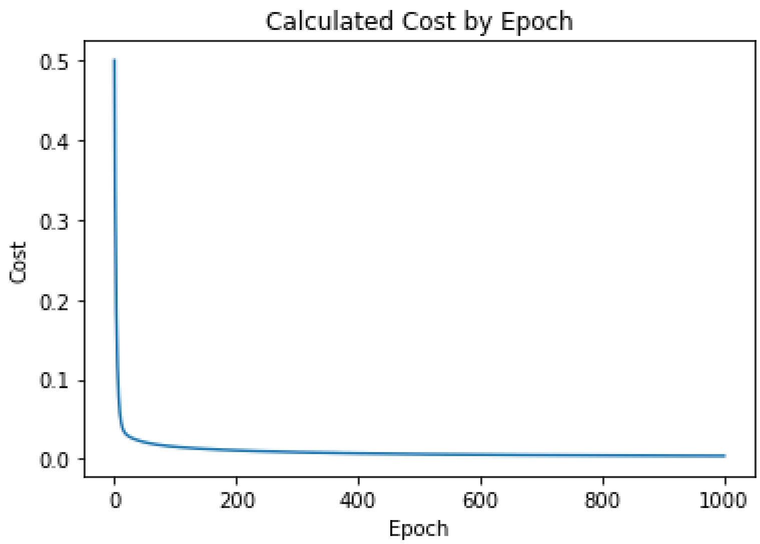
	Coefficients
Pixel number	
0	0.000000
1	0.000000
2	0.000000
3	0.000000
4	0.000000
...	...
780	0.000000
781	0.000000
782	0.000000
783	0.000000
784	0.000024

785 rows × 1 columns

In [21]:
```
######### Gradient Descent Implementation #########
######################################################
# initialize the weights
b = np.zeros(shape=len(train_X.columns))

# this will hold the calculated cost at each epoch(iteration)
cost_history = []

# calculate the partial derivative with respect to b_i and the cost in each epoch
for i in range(1000):
    # calculates y hat (train)
    train_yhat = train_X.dot(b)

    # calculate the partial derivative with respect to b_i
    Db = (-2/len(train_X))*((train_y-train_yhat).dot(train_X))

    # small step size to prevent coefficients from increasing to positive infinit
    b-= 0.0000001*Db

    current_cost = cost(train_yhat,train_y)
    # Save the cost history at each iteration
    cost_history.append(current_cost)

    # display iteration number and cost at each iteration.
    print("iteration",str(i) + ":  Cost =",current_cost)
b_opt=b
```

```
iteration 0:  Cost = 0.5
iteration 1:  Cost = 0.33732019036017025
iteration 2:  Cost = 0.24678885829075772
iteration 3:  Cost = 0.18730853272299994
iteration 4:  Cost = 0.14552366257172883
iteration 5:  Cost = 0.11545105271461722
iteration 6:  Cost = 0.09359485536223242
iteration 7:  Cost = 0.07761876812289532
iteration 8:  Cost = 0.06588010235292362
iteration 9:  Cost = 0.057204018817377356
iteration 10:  Cost = 0.05074553605344117
iteration 11:  Cost = 0.0458956729613355
iteration 12:  Cost = 0.042215099441032876
iteration 13:  Cost = 0.039386636282379456
iteration 14:  Cost = 0.03718111182498403
iteration 15:  Cost = 0.03543278717992985
iteration 16:  Cost = 0.03402165996340251
iteration 17:  Cost = 0.03286071823226338
iteration 18:  Cost = 0.03188675823445167
```

In [22]:
```
# Display a graph of Cost Vs Epoch
plt.plot(cost_history)
plt.title("Calculated Cost by Epoch")
plt.xlabel("Epoch")
plt.ylabel("Cost")
plt.show()
```



In [28]:

```
# prints out the coefficients:
coefficients_GD=[]
print("Coefficients:")
for (pixel,coef) in dict(zip(train_X.columns,b_opt)).items():
    coefficients_GD.append(coef)

# classifying all predicted values above .5 as 1 and below .5 as zero (train)
train_yclass = (train_X.dot(b_opt) > .5).values
print("Training Classification Accuracy:",np.mean(train_yclass == train_y))

# classifying all predicted values above .5 as 1 and below .5 as zero (test)
test_yclass = (test_X.dot(b_opt) > .5).values
print("Test Classification Accuracy:",np.mean(test_yclass == test_y))

coefficients_GD=pd.DataFrame(coefficients_GD)
coefficients_GD.columns=["Coefficients"]
coefficients_GD.index.names=["Pixel number"]

coefficients_GD
```

```
Coefficients:
Training Classification Accuracy: 1.0
Test Classification Accuracy: 0.99
```

Out[28]:

	Coefficients
Pixel number	
0	0.000000
1	0.000000
2	0.000000
3	0.000000
4	0.000000
...	...
780	0.000000
781	0.000000
782	0.000000
783	0.000000
784	0.000003

785 rows × 1 columns